Theoretical Computer Science 68 (1989) 19-36
North-Holland

EFFICIENT SIMULATIONS OF SIMPLE MODELS OF PARALLEL COMPUTATION BY TIME-BOUNDED ATMs AND SPACE-BOUNDED TMs*

Jik H. CHANG and Oscar H. IBARRA

Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, U.S.A.

Michael A. PALIS

Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104-3897, U.S.A.

Communicated by M. Harrison
Received March 1986
Revised July 1988

Abstract. We present efficient time-bounded ATM and space-bounded TM simulations of one-way conglomerates (OWCs), which are interconnected networks of finite-state machines that allow only one-way communication between adjacent nodes. In particular, we show that OWCs with depth $D(n)$ and operating in time $T(n)$ can be simulated by ATMs in time $O(D(n) \cdot \log T(n))$ (and hence by a TM with the same amount of space). This extends Ruzzo's result that boolean circuits of depth $D(n)$ can be simulated by $O(D(n))$-time bounded ATMs, and refines Goldschlager's result that two-way conglomerates operating in $T(n)$ time can be simulated by $T(n)$-space bounded TMs. By exploiting the regularity of interconnections in some OWCs, we obtain more efficient space-bounded TM simulations. For example, using the ATM result, a k-dimensional one-way mesh array of n^k-nodes would require n^2 space on a TM (such an array can run in c^n time in the worst case). We show that the space can be reduced to $n^{2-1/k}$.

1. Introduction

In this paper, we study the complexity of languages recognizable by arrays of finite-state machines of various interconnections in terms of Turing machines (TMs) and alternating Turing machines (ATMs) [2, 20, 21, 24]. Examples are tree arrays [1, 5, 8, 9, 10, 23], triangular arrays [6, 12] and k-dimensional mesh-connected arrays [7, 13, 15, 18, 19, 22].

Space-bounded TM simulations of such arrays have been studied in [11], where they were referred to as *conglomerates*. Conglomerates allow *two-way* communication between adjacent processors. In [11], it was shown that a conglomerate operating in $T(n)$ parallel time can be simulated by a TM in space $T(n)$. Goldschlager [11] did not have a separate treatment for the special case of *one-way* conglomerates, which only allow one-way communication between processors. Consider, e.g. a 2-dimensional one-way mesh-connected array (2-OWMA) of size $n \times n$. The input is applied in parallel at time 0 to the nodes at positions $(i, 0)$, $0 \leq i < n$,

* This research was supported in part by NSF Grants MCS 83-04756, DCR-8420935 and DCR-8604603.

and the output node is at position $(n-1, n-1)$. There are unidirectional edges from node (i,j) to nodes $(i+1,j)$ and $(i,j+1)$. Such an array can solve a variety of problems, e.g. it can implement dynamic programming and transitive closure algorithms in linear time [6, 12, 16, 19]. A 2-OWMA operating in $O(n^2)$ time would need n^2 space on a TM if we use the simulation in [11], which is what one obtains also by direct simulation.

It seems reasonable to suspect that better TM simulations of one-way conglomerates can be found since they are simple generalizations of combinational (or boolean) circuits whose gates have memory [21]. However, unlike a combinational circuit which operates in time equal to its depth, a one-way conglomerate can use its memory to operate in time greater than its depth. In [21], it was shown that (uniform) boolean circuits of depth $D(n)$ can be simulated by ATMs in time $O(D(n))$ and hence by TMs in space $D(n)$ (since a $T(n)$-time bounded ATM can be simulated by a $T(n)$-space bounded TM [2]).

In this paper, we refine and extend the results in [11] and [21], respectively, by giving efficient simulations of one-way conglomerates by time-bounded ATMs and space-bounded TMs. In particular, we show that a one-way conglomerate with depth $D(n)$ and operating in time $T(n)$ can be simulated by an ATM in time $O(D(n) \cdot \log T(n))$,[1] and hence by a TM using the same amount of space. Thus, for example, a 2-OWMA operating in $T(n)$ time can be simulated by an $n \log T(n)$-space bounded TM, which is better than that obtained using the result in [11], or the straightforward simulation for the case when $T(n) = 2^{o(n)}$. We obtain similar results for other one-way conglomerates. For example, we prove that a one-way $O(n)$-node tree array (with the input fed in parallel to the leaves and the root as the output node) can only accept languages in $NC^{(2)}$ (= class of languages recognizable by uniform boolean circuits with a polynomial number of gates and depth $O(\log^2 n)$ [21]). This is in contrast with the fact that a two-way tree array is equivalent to a n-space bounded TM.

By exploiting the regularity of interconnections in some one-way conglomerates, more efficient space-bounded TM simulations can be found by using the TM directly. For example, from the above discussion, a 2-OWMA (which can be shown to run in c^n time in the worst-case) would require n^2 TM space. On the other hand, we give a better TM simulation that requires only $n\sqrt{n}$ space. This result extends to higher-dimensions: an n^k-processor bounded k-OWMA (which can be shown to run in c^n worst-case time) can be simulated by a TM in space $n^{2-1/k}$. Similar results are obtained for other one-way conglomerates, e.g. tree arrays and triangular arrays. Finally, as corollaries, we also show that two-way arrays are strictly more powerful than their one-way counterparts.

The paper is organized as follows. Section 2 defines one-way conglomerates as well as some specific examples, e.g. tree arrays, triangular arrays and k-dimensional mesh-connected arrays. Section 3 gives the general simulation result of one-way

[1] In this paper, all logarithms are base 2.

conglomerates by time-bounded ATMs. This section also answers some of the open problems posed in [25]. Section 4 gives the space-bounded TM simulations for some specific arrays. Here, it is also shown that two-way arrays are strictly more powerful than one-way arrays. Finally, Section 5 ends the paper with some concluding remarks.

2. One-way conglomerates

A *one-way conglomerate* (OWC) is a directed acyclic graph, where each node represents a finite-state machine. Each node has indegree at most d, for some integer constant $d \geq 0$. Nodes with the same indegree k, $0 \leq k \leq d$, represent the same finite-state machine and are called type-k nodes. Exactly n nodes are type-0, henceforth referred to as *input* nodes. We assume that these nodes are numbered $1, 2, \ldots, n$. The nodes can have unbounded outdegree; exactly one node has outdegree 0 and is called the *output* (or *accepting*) node. We assume that every non-output node has at least one directed path from itself to the output node.

At time 0, all nodes (except the input nodes) are in a quiescent state q_0. The input $a_1 a_2 \ldots a_n$ is applied to the OWC at time 0 by setting the state of input node i to a_i. The next state of a node is a function of its current state and the current states of all its immediate predecessors; for a type-k node, this is given by the transition function $\delta_k : Q^{k+1} \to Q$. The size $Z(n)$ of the OWC is the number of nodes; the *depth* $D(n)$ is the length of a longest path from an input node to the output node. The OWC *accepts* a string $x = a_1 a_2 \ldots a_n$ iff the output node enters an accepting state at any time $t \geq D(n)$. It has *time complexity* $T(n)$ iff any input of length n that is accepted requires no more than $T(n)$ time steps to accept. Because the nodes have bounded indegree, we have the following relationship between the complexity measures: $T(n) \geq D(n) = \Omega(\log Z(n)) \geq \Omega(\log n)$.

A OWC is on the one hand, a special case of a conglomerate with one-way communication between processors, and on the other hand, a generalization of a combinational (boolean) circuit [21] whose gates have memory. Note that, unlike a combinational circuit which operates in time equal to its depth, a OWC can use its memory to operate in time t greater than its depth.

Unlike other machine models where a single machine handles inputs of all sizes (e.g. TMs), one must construct a distinct OWC for each input length. Thus, a language L is, in fact, associated with a family $C = (c_1, c_2, \ldots)$ of OWCs, where c_n is the OWC for strings of length n.

In order to describe each OWC in the family C, some sort of encoding is necessary. We adopt the encoding given in [21] (for combinational circuits) by associating with the family C a *connection language* which describes, for each c_n in C, the manner in which the nodes are interconnected.

Following [21], we assume that for any family $C = (c_1, c_2, \ldots)$ of OWCs, the nodes of c_n are numbered so that node 0 is the output node and nodes $1, 2, \ldots, n$

are the input nodes. Also, we restrict the node numbering so that the largest node number is $(Z(n))^{O(1)}$, where $Z(n)$ is the size of c_n. Thus, the node numbers, coded in binary, have length $O(\log Z(n))$. Moreover, we assume that the predecessors of a node are ordered so that we can refer to the first, second, etc., predecessors of the node.

Let d be the maximum indegree of a node in any $c_n \in C$. Let $D = \{1, 2, \ldots, d\}$. If x is a node in c_n and $p \in D^*$, let $x(p)$ denote the node reached by following the path p of predecessors to x. That is, $x(\varepsilon)$ is x, $x(1)$ is x's first predecessor, $x(13) = x(1)(3)$ is x's first predecessor's third predecessor, etc.

Definition 2.1. The *extended connection language* L_{EC} of the family C is the set of strings of the form $\langle n, x, p, y\rangle$ such that n and $x \in \{0,1\}^*$, $p \in D^*$, $y \in \{0,1\}^* \cup \{\text{type-0}, \text{type-1}, \ldots, \text{type-}d\}$, and in c_n either

(i) $p = \varepsilon$ and x is a node of type y, $y \in \{\text{type-0}, \text{type-1}, \ldots, \text{type-}d\}$, or

(ii) $|p| \leq \log Z(n)$ and $x(p)$ is numbered y, $y \in \{0,1\}^*$.

Thus, L_{EC} encodes the name (or number) and type of each node, as well as the names of its predecessors within a distance $\log Z(n)$ away. We shall use L_{EC} later in defining *uniform* families of one-way conglomerates.

We end this section by giving some examples of OWCs. Figure (1a) pictures a OWC called a *one-way tree array* (OWTA), in which the nodes are connected as a full binary tree. All edges are directed towards the root, which is the accepting node. The input $a_1 a_2 \ldots a_n$ is applied in parallel to the leaves at time 0 by setting the states of the leaves to $a_1 a_2 \ldots a_n \$^k$ from left to right, where k is the smallest nonnegative integer such that $n + k$ is a power of 2. OWTAs have been studied by Dyer and Rosenfeld (which they referred to as UTCAs) [10]; they can accept some very interesting languages, even when the computing time is restricted to $O(\log n)$, $O(\log^2 n)$, $O(n)$, etc. (see [10]). Note that a OWTA has depth $\lceil \log n \rceil$.

Figure 1(b) illustrates a *one-way triangular array* (OWTRA) whose accepting node is the node at the apex. Clearly, a OWTRA has depth $n-1$. OWTRAs have been shown to solve a wide variety of problems, especially those that can be solved via dynamic programming. In particular, the Earley and Cocke-Kasami-Younger algorithms for context-free language recognition can be implemented by OWTRAs operating in linear time and using n^2 processors [6, 12, 16, 19].

Figure 1(c) shows an example of a *k-dimensional one-way mesh array* (k-OWMA), for the case $k = 2$. The nodes of a k-OWMA occupy the points of a k-dimensional grid. The k-OWMA is said to be $S^k(n)$-processor bounded iff each of the k dimensions of the array has $S(n) \geq n$ processors. If the node at the origin is given position $(0, 0, \ldots, 0)$, then the accepting node is at position $(S(n)-1, S(n)-1, \ldots, S(n)-1)$. Hence, the depth is $O(S(n))$. The input $a_1 a_2 \ldots a_n$ is applied to the k-OWMA at time 0 by setting the state of the node at position $(i, 0, \ldots, 0)$, $0 \leq i < n$, to a_i. All other nodes are initially in a quiescent state. (Technically, a k-OWMA is not a one-way conglomerate since, except for the node at position $(0, 0, \ldots, 0)$,

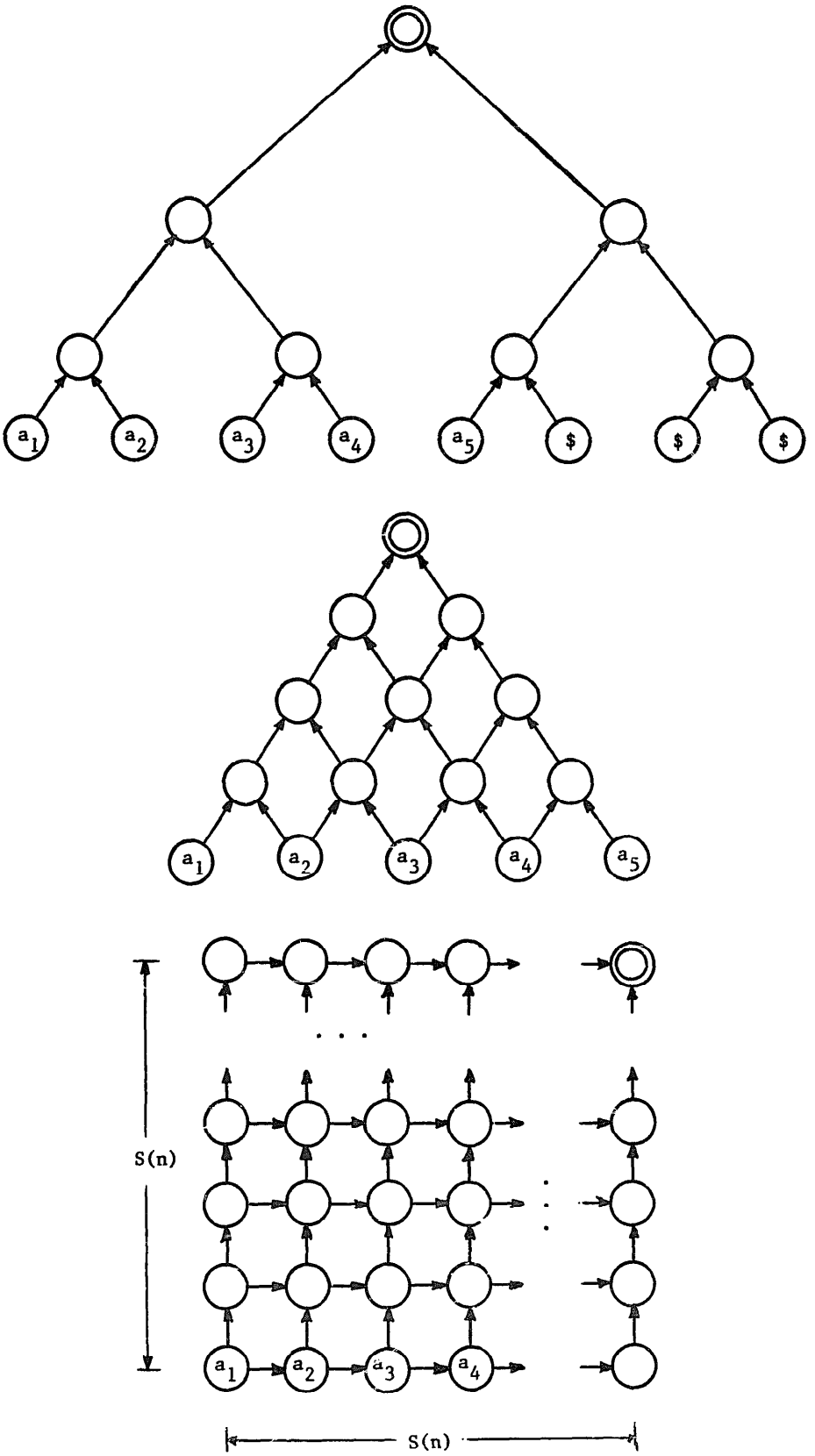


Fig. 1. Some examples of OWCs. (a) OWTA; (b) OWTRA; (c) k-OWMA for $k=2$.

the nodes receiving input symbols do not have indegree zero. However, it can be made to conform with the definition of OWC by introducing n extra nodes of indegree zero to hold the input symbols.)

One-way conglomerates with other interconnection patterns can also be defined. However, in this paper we restrict our attention to the three types defined above (as well as a few others to be defined later).

3. Simulations by time-bounded alternating Turing machines

The result in [11] states that conglomerates (and hence one-way conglomerates) operating in $T(n)$ parallel time can be simulated by a Turing machine in space $T(n)$. This result is rather general and one might suspect that a better simulation can be obtained for the special case of OWCs. In this section we show that this is indeed the case; in fact, the simulation can be carried out by time-bounded alternating Turing machines [2, 20, 21, 24]. That time-bounded ATM simulations are (probably) better than space-bounded TM simulations follows from the fact that a $T(n)$-time bounded ATM can be simulated by a $T(n)$-space bounded TM [2]. The converse is open, although it is known that a $T(n)$-space bounded TM can be simulated by a $T^2(n)$-space bounded ATM [2, 20].

Before stating our result, we first give an informal definition of an ATM (see [2] for a formal definition) and define the concept of *uniform* families of OWCs.

An alternating Turing machine [2, 20, 21, 24] is a generalization of a nondeterministic TM whose state set is partitioned into "universal" and "existential" states. As with a nondeterministic TM, we can view the computation of an ATM as a tree of configurations. A configuration is called universal (existential) if the state associated with the configuration is universal (existential). A computation tree of an ATM M on input w is a tree whose nodes are labeled by configurations of M on w, such that the root is the initial configuration and the children of any non-leaf node labeled by a universal (existential) configuration include all (one) of the immediate successors of that configuration. A computation tree is accepting if it is finite and all the leaves are accepting configurations. M accepts w if there is an accepting computation tree for M on input w. Note that nondeterministic TMs are essentially ATMs with only existential states. We assume that ATMs have a read-only input tape with endmarkers. Unlike [2], we will not consider "negating states" here.

We will use a variant of an ATM, called an indexing ATM [20], which allows sublinear time bounds. An indexing ATM has a special "index tape"; whenever an integer i is written on the index tape, the ith symbol of the input is immediately accessible to the ATM. Thus, in $\log n$ steps, it can read any position on the input tape. Unless otherwise stated, an ATM is assumed to be an indexing ATM.

An ATM has time complexity $T(n)$ if for all accepted inputs of length n, there is an accepting computation tree of height at most $T(n)$. Similarly, an ATM has space complexity $S(n)$ if for all accepted inputs of length n, there is an accepting

computation tree each of whose nodes is labeled by a configuration whose space is at most $S(n)$.

Recall that the extended connection language L_{EC} (Definition 2.1) of a family C of OWCs encodes the information about how nodes of each conglomerate c_n in the family are interconnected. In order that C can be efficiently simulated by an ATM, it is necessary to impose the restriction that the family be *uniform*, i.e. its associated L_{EC} can be efficiently recognized by an ATM. More precisely, we have the following.

Definition 3.1. Let $C=(c_1, c_2, \dots)$ be a family of OWCs. Furthermore, let $D(n)$ and $Z(n)$ be the depth and size, respectively, of c_n. Then C is said to be *uniform* iff there is an ATM recognizing L_{EC} which on inputs of the form $\langle n, -, -, -\rangle$ takes $O(D(n))$ time and $O(\log Z(n))$ space.

We are now ready to prove the following.

Theorem 3.1. *Let C be a uniform family of OWCs and $D(n)$, $Z(n)$ and $T(n)$ be the depth, size and time complexity, respectively, of c_n. Then the language accepted by C can be accepted by an* ATM *M in time $O(D(n)\cdot \log T(n))$. Moreover, the space used by M is $O(\max\{\log Z(n), \log T(n)\})$.*

Proof. The heart of the simulation is carried out by procedure CHECK($[n, x, p], [q_1, t_1], [q_2, t_2]$), which verifies that node $x(p)$ of c_n can go from state q_1 at time t_1 to state q_2 at time t_2. For ease of manipulation, we assume that each of the three arguments are on separate tapes.

Let M' be an $O(D(n))$-time bounded and $O(\log Z(n))$-space bounded ATM accepting L_{EC}. Then M simulates c_n of the family C as follows:

(I) M guesses and writes n in binary, and verifies it by checking that the $(n+1)$st position on the input tape is the endmarker. Next M guesses $Z(n)$ and marks an area of $\log Z(n)$ cells on the tape for the argument $[n, x, p]$ to hold the value of p. (This is used to check in one step whether $|p|=\log Z(n)$.)

(II) M guesses and writes $t \leq T(n)$ in binary, guesses a final state q_f of the accepting node of c_n, and executes CHECK($[n, 0, \varepsilon], [q_0, 0], [q_f, t]$).

Procedure CHECK($[n, x, p], [q_1, t_1], [q_2, t_2]$);
//Verifies that node $x(p)$ of c_n can go from state q_1 at time t_1 to state q_2 at time t_2.//
(1) If $t_2=t_1+1$, then do the following; otherwise, go to step (2).
  (1.1) If $|p|=\log Z(n)$, guess an integer y, then split at a universal state doing both steps (1.1.1) and (1.1.2); otherwise, go to step (1.2).
    (1.1.1) Check that $y=x(p)$ by running M' on $\langle n, x, p, y\rangle$.
    (1.1.2). Set x to y, p to ε, and go to step (1.2).
  (1.2) Guess the type-$k \in$ {type-0, type-1, ..., type-d} of node $x(p)$, then split at a universal state doing step (1.2.1) and either step (1.2.2) or step (1.2.3) (depending on the value of k).
    (1.2.1) Check that $x(p)$ is type-k by guessing y, then running M' on both $\langle n, x, p, y\rangle$ and $\langle n, y, \varepsilon,$ type-$k\rangle$.

(1.2.2) If $x(p)$ is type-0 (input node), check that $\delta_0(q_1) = q_2$. If $t_1 = 0$, guess a y, $1 \leq y \leq n$, then universally (i) check that $x(p) = y$ by running M' on $\langle n, x, p, y \rangle$, and (ii) read input symbol a_y and halt, accepting iff $q_1 = a_y$.

(1.2.3) If $x(p)$ is type-k, $k \neq 0$, then proceed as follows: If $t_1 = 0$, check that $q_1 = q_0$. For each predecessor $x(p \cdot i)$ of $x(p)$, $1 \leq i \leq k$, guess its state r_i at time 0 and its state s_i at time t_1. Check that $\delta_k(q_1, s_1, \ldots, s_k) = q_2$. Then, splitting at a universal state, execute for all $1 \leq i \leq k$ CHECK($[n, x, p \cdot i]$, $[r_i, 0]$, $[s_i, t_1]$).

(2) If $t_2 > t_1 + 1$, guess the state q of $x(p)$ at time $t = [(t_1 + t_2)/2]$. Then verify the guess by splitting at a universal state and executing CHECK($[n, x, p]$, $[q_1, t_1]$, $[q, t]$) and CHECK($[n, x, p]$, $[q, t]$, $[q_2, t_2]$).

end CHECK.

It is easy to verify that M uses space $O(\max\{\log Z(n), \log T(n)\})$. For the time complexity, we first note that the computation is divided into $D(n)$ phases, where each phase starts with a recursive call to procedure CHECK (step (II) for the first phase and step (1.2.3) for all others). Each phase then continues with $O(\log T(n))$ executions of step (2) and ends with an execution of step (1).

Consider now the time contributions of the steps executed within each phase. A straightforward implementation of step (2) would require $O(\log T(n))$ steps per execution since a new time-stamp t has to be guessed and written (the first argument $[n, x, p]$ does not change). Thus, $O(\log T(n))$ executions of this step would require $O(\log^2 T(n))$ time. We show later that this can be reduced to $O(\log T(n))$.

For the remaining steps, we first observe that steps (1.1.1), (1.2.1) and (1.2.2) are not recursive and hence each contribute at most an additive term to the *total* running time of M. Steps (1.1.1) and (1.2.1) take $O(D(n))$ time (to run M') and $O(\log Z(n) + D(n))$ time (to guess y and run M'), respectively. Similarly, step (1.2.2) takes $O(\log n + D(n))$ time (to guess y, $1 \leq y \leq n$, and execute (i) and (ii) in parallel). Thus, the contribution of steps (1.1.1), (1.2.1) and (1.2.2) to the total running time of M is at most an additive term of $O(D(n))$ (since $D(n) = \Omega(\log Z(n)) \geq \Omega(\log n)$).

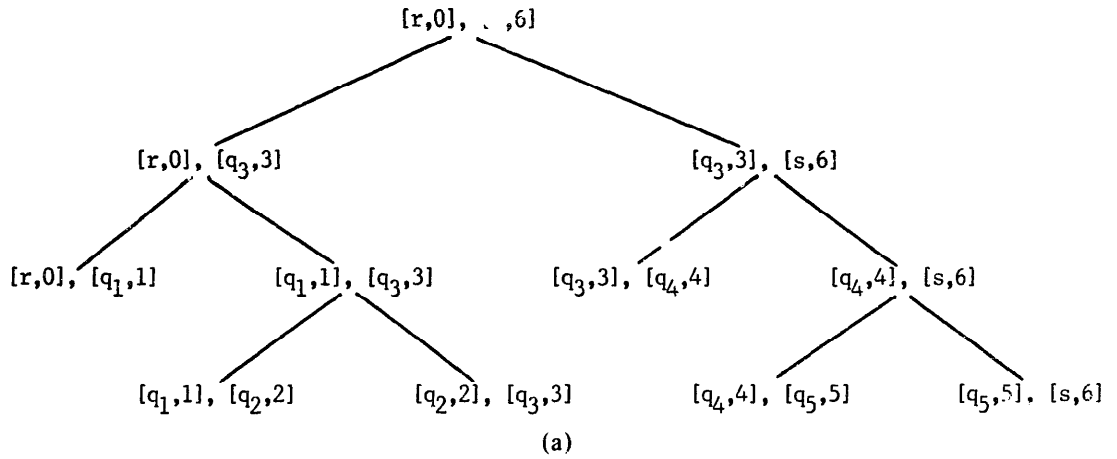
Step (1.1) (minus step (1.1.1)) takes $O(\log Z(n))$ time whenever $|p| = \log Z(n)$ and takes $O(1)$ time otherwise. Since $|p|$ becomes equal to $\log Z(n)$ at intervals of $\log Z(n)$ consecutive phases, step (1.1) averages $O(1)$ time per phase.

Step (1.2) (minus steps (1.2.1) and (1.2.2)) takes $O(\log T(n))$ time to set up the new time-stamps for the new recursive calls to CHECK in step (1.2.3) and $O(1)$ time on all others (i.e. to guess the type of $x(p)$, the predecessors' states r_i and s_i, to write $x(p \cdot i)$, etc.).

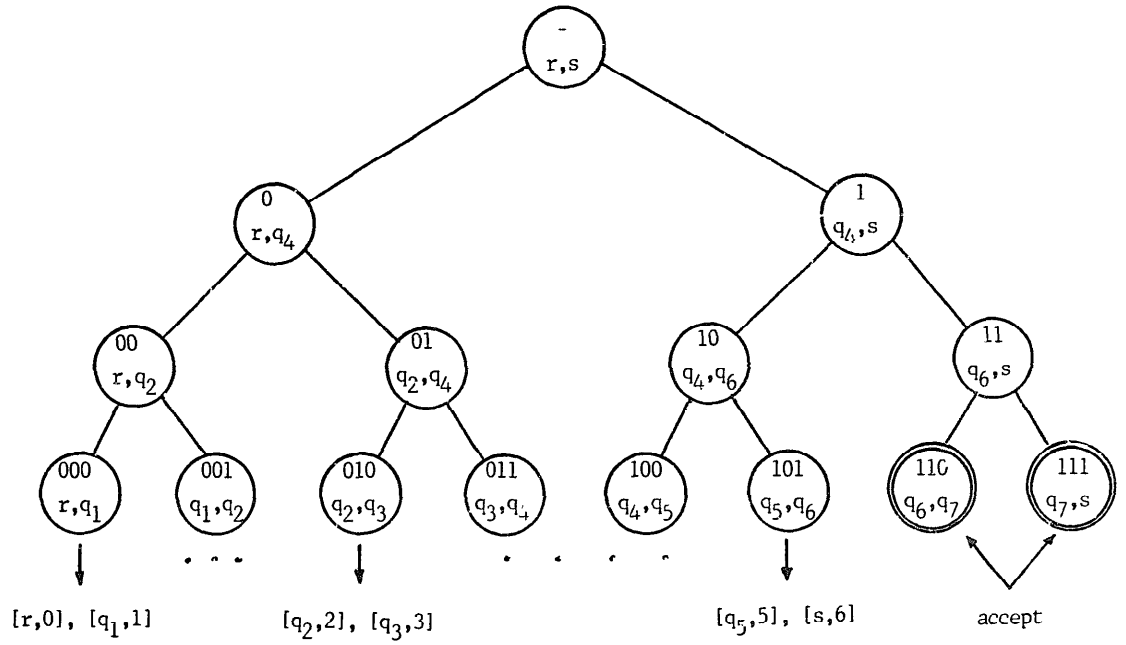
Thus, each phase is completed in $O(\log T(n))$ steps. The total contribution of the $D(n)$ phases is $D(n) \cdot O(\log T(n))$. Counting the time spent in steps (I) and (II) (which is $O(\log n + \log T(n) + \log Z(n)) = O(\log T(n) + D(n))$) plus the additive term due to steps (1.1.1), (1.2.1) and (1.2.2) (which is $O(D(n))$, the total running time is thus $O(D(n) \cdot \log T(n))$.

Consider now the O(log $T(n)$) executions of step (2) during each phase. First observe that this step is first executed when the argument to CHECK is of the form $([n, x, p], [r, 0], [s, t])$. The difference between the first and second time-stamps is then recursively halved until the difference is one. The portion of the computation tree associated with this recursion is a binary tree of depth log t and exactly t leaves, as shown in Fig. 2(a) for the case $t = 6$. (We have omitted the argument $[n, x, p]$ since this does not change during the recursion.) Because M has to write new pairs of time-stamps at each recursive call, the actual time spent by M is $O(\log^2 t)$.

M can avoid explicitly writing the new time-stamps at each level of the tree by using the length of log t to calculate the depth of recursion. M does this by "building" a full binary tree of depth $d = \lfloor \log t \rfloor + 1$ (the length of the binary representation of t) as shown in Fig. 2(b). At each node, M writes the labels of



(a)



(b)

Fig. 2.

the edges traversed from the root to the node, with a left and right edge having label 0 and 1, respectively. This takes O(1) time per recursive level. Thus, the 2^d leaves of the tree would be labelled $0, 1, \ldots, 2^d - 1$. From among these, exactly t leaves with the required pairs of time-stamps can be chosen as follows. For each leaf labelled v, M computes the difference $t - v$. If this difference is less than or equal to zero, M halts and enters an accepting state. For each of the remaining t leaves, M creates a pair of time-stamps such that the first time-stamp is the label of the leaf and the second time-stamp is the first time-stamp plus one.

At each node of the tree, M also guesses a state (see Fig. 2(b)). More precisely, given a pair of states (q_1, q_2) at a node, M guesses a new state q and passes the pair (q_1, q) to the node's left child and the pair (q, q_2) to the node's right child. At the root, M initially has the given pair of states (r, s) (of node $x(p)$ of c_n) at times 0 and t, respectively. At each leaf, the first and second states associate with the first and second time-stamps, respectively. Thus, for the leftmost leaf the first [state, time-stamp] = [r, 0] and, for every two consecutive leaves l_1 and l_2, the second [state, time-stamp] of l_1 = the first [state, time-stamp] of l_2, as should be the case. Finally, the second [state, time-stamp] of the rightmost nonaccepting leaf is correct except for the state; the time-stamp = t as it is supposed to be, but the state, since guessed, may not equal s, the given state at time t. This is easily remedied by replacing [state, t] in the leftmost nonaccepting leaf by [s, t]. (This is done by remembering s as M goes down the tree; the leftmost nonaccepting leaf can be identified as it is the only leaf with label v such that $t - v = 1$.)

Since the [state, time-stamp] pairs are constructed only at the leaves, it is clear that the depth of the computation tree corresponding to the O(log $T(n)$) executions of step (2) during each phase is O(log $T(n)$). □

Remark 3.1. The above result extends to OWCs whose input nodes i, $1 \leq i \leq n$, each has a serial input $a_{i,1} \ldots a_{i,n}$ of length n, instead of a single symbol. In this case, input node i is at a quiescent state q_0 at time 0 and its state at time $t > 0$ is a function of the state at time $t-1$ and the input symbol $a_{i,t}$ at time $t \leq n$ (the input is λ for all $t > n$). The simulation by the ATM M remains the same except that its input consists of the n input strings, and step (1.2.2) of procedure CHECK is modified to:
Step (1.2.2'): If $x(p)$ is type-0, proceed as follows. Set $a = \lambda$. If $0 < t_1 \leq n$, guess a y, $1 \leq y \leq n$, then split at a universal state doing both (i) and (ii); otherwise, proceed to (iii).

(i) Check that $x(p) = y$ by running M' on $\langle n, x, p, y \rangle$.
(ii) Read input symbol a_{y,t_1}, set $a = a_{y,t_1}$, then go to step (iii).
(iii) If $t_1 = 0$, check that $q_1 = q_0$. Check that $x(p)$ can go in one step from state q_1 to state q_2 while reading input symbol a.

It can be verified that OWTAs, OWTRAs and k-OWMAs form uniform families of one-way conglomerates in the sense of Definition 3.1. Thus, we have the following corollary to Theorem 3.1.

Corollary 3.1.

(1) *Every language accepted by a $T(n)$-time bounded* OWTA *can be accepted by an ATM in time* $O(\log n \log T(n))$ *and space* $O(\max\{\log n, \log T(n)\})$.

(2) *Every language accepted by a $T(n)$-time bounded* OWTRA, *can be accepted by an* ATM *in time* $O(n \log T(n))$ *and space* $O(\max\{\log n, \log T(n)\})$.

(3) *Every language accepted by a $T(n)$-time bounded and $S^k(n)$-processor bounded k-*OWMA *can be accepted by an* ATM *in time* $O(S(n) \log T(n))$ *and space* $O(\max\{\log S(n), \log T(n)\})$.

We now show that the running time of a OWC cannot be arbitrarily large.

Lemma 3.1. *Let A be a* OWC *with depth $D(n)$. If A enters an accepting state, then it does so in no more than $c^{D(n)}$ time steps, for some constant $c > 0$.*

Proof. Define the depth of a node x of A as the length of a longest path from an input node to x. Thus, an input node has depth zero and the accepting node has depth $D(n)$. We prove the lemma by showing that a node x of A at depth d enters a sequence of states which has a period $\leq c^{d+1}$ for some constant $c > 0$.

Let s be the maximum cardinality of the state sets of the nodes of A. Without loss of generality, assume the $s \geq 2$. First observe that any integer t, $0 < t \leq s$, can be expressed as

$$t = \prod_{j=1}^{r} p_j^{e_j},$$

where the p_js are all the prime numbers less than or equal to s and $0 \leq e_j < s$ for $1 \leq j \leq r$.

We prove by induction on the depth d of a node of A that

(*) A node at depth d enters a sequence of states with a period

$$\rho = \prod_{j=1}^{r} p_j^{u_j},$$

such that $0 \leq u_j < s \cdot (d+1)$ for $1 \leq j \leq r$.

Induction base: For $d = 0$, the node is an input node and hence enters a sequence of states with a period $t \leq s$. It follows that (*) is true for $d = 0$.

Induction step: Assume that (*) is true for all nodes at depth $< d$. Let x be a node at depth d. If x is a type-k node, then it has k predecessors x_i, $1 \leq i \leq k$, such that x_i is at depth $d_i < d$. By the induction hypothesis, predecessor x_i enters a sequence of states with a period

$$\rho_i = \prod_{j=1}^{r} p_j^{v_{i,j}},$$

such that $0 \leq v_{i,j} < s \cdot (d_i + 1) \leq s \cdot d$ for $1 \leq j \leq r$.

It is easy to see that the period ρ of node x must be $\rho = t \cdot \mathrm{LCM}(\rho_1, \rho_2, \ldots, \rho_k)$, for some $0 < t \leq s$, where $\mathrm{LCM}(\rho_1, \rho_2, \ldots, \rho_k)$ is the least common multiple of $\rho_1, \rho_2, \ldots, \rho_k$. Thus,

$$\rho = t \cdot \prod_{j=1}^{r} p_j^{u_j} = \prod_{j=1}^{r} p_j^{u_j + e_j},$$

where $u_j = \max_{1 \leq i \leq k}\{v_{i,j}\} < s \cdot d$ and $0 \leq e_j < s$, for $1 \leq j \leq r$. It follows that $u_j + e_j < s \cdot (d+1)$ for $1 \leq j \leq r$. Thus, $(*)$ is true for $d > 0$.

$(*)$ implies that for a node at depth d the period is no more than c^{d+1} for some constant $c > 0$. Moreover, the nonperiodic prefix (if any) of the sequence of states is no longer than c^{d+1}. Thus, if the accepting node of A (which is at depth $D(n)$) enters an accepting state, it does so in time no more than $c_1^{D(n)}$ for some constant $c_1 > 0$. □

Remark 3.2. An OWC, even a 1-OWMA, can actually take time $c^{D(n)}$ to accept.

From Corollary 3.1 and Lemma 3.1, we have the following.

Corollary 3.2.

(1) *Every language accepted by a* OWTA *can be accepted by an* ATM *in time* $O(\log^2 n)$ *and space* $O(\log n)$.

(2) *Every language accepted by a* OWTRA *can be accepted by an* ATM *in time* $O(n^2)$ *and space* $O(n)$.

(3) *Every language accepted by a* $S^k(n)$*-processor bounded* k-OWMA *can be accepted by an* ATM *in time* $O(S^2(n))$ *and space* $O(S(n))$.

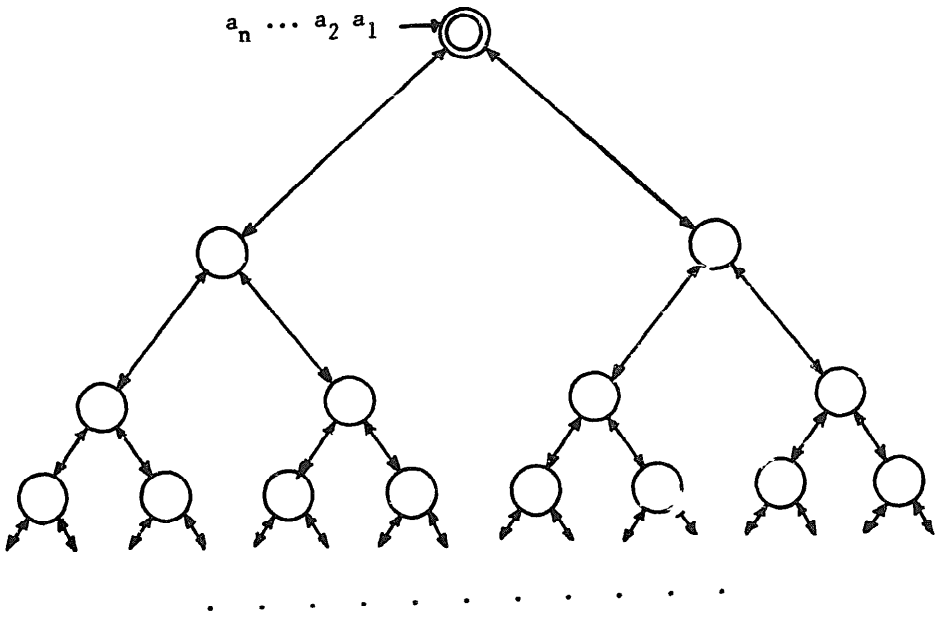
In particular, Corollary 3.2(1) states that every language accepted by a OWTA is in $\mathrm{NC}^{(2)}$ (= class of languages recognizable by uniform boolean circuits with a polynomial number of gates and depth $O(\log^2 n)$ [20].

We can define *two-way* versions of OWTAs, OWTRAs and k-OWMAs by replacing each unidirectional edge by a bidirectional edge connecting the same pair of nodes. (We name the two-way array just like the corresponding one-way array, except that the letter "O" is replaced by "T" for "two-way"). For two-way arrays, Theorem 3.1 does not apply because the underlying graphs are no longer acyclic. However, one can still obtain an ATM simulation by making use of the "unrolling" technique. The idea is to view the computation of the two-way array A as a combinational circuit C consisting of memoryless gates $\{\langle x, t\rangle \mid x$ is a node of A and $t \geq 0\}$. Gate $\langle x, t\rangle$ is connected to gate $\langle y, t+1\rangle$ iff $x = y$ or x is adjacent to y in A. Each gate computes the δ function of the corresponding node of A so that the output of $\langle x, t\rangle$ is the state of node x at time step t. Clearly, C has no cycles. Moreover, if A has time complexity $T(n)$, then C has depth $T(n)$. Thus, using the techniques in [11, 21], A can be simulated by a $O(T(n))$-time bounded ATM. Thus, we have the following.

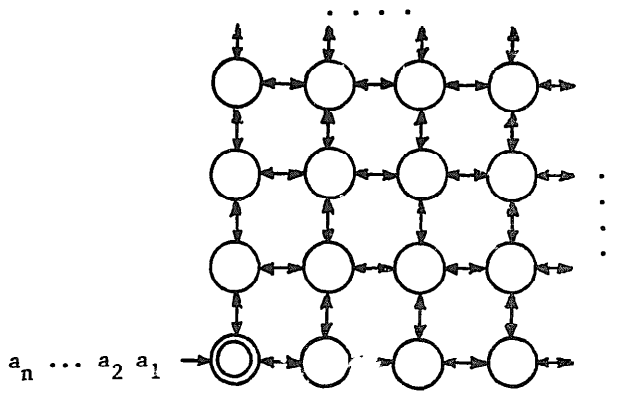
Corollary 3.3. *Every language accepted by a $T(n)$-time bounded* TWTA, TWTRA, or k-TWMA *can be accepted by an* ATM *in time* $O(T(n))$.

Finally, we consider two-way arrays with an unbounded number of processors. Figure 3(a) shows an *iterative tree array* (ITA) whose nodes are connected as an infinite full binary tree. The input is fed serially at the root, which is also the accepting node. Figure 3(b) shows an example of a *k-dimensional iterative mesh array* (k-IMA), in which each dimension can have an infinite number of nodes. The input is fed serially to the node at the origin, which is also the accepting node.

For $T(n)$-time bounded ITAs or k-IMAs, the unrolled computations result in combinational circuits of depth $O(T(n))$ so that Corollary 3.3 also applies to these array models.



(a)



(b)

Fig. 3. Unbounded two-way arrays. (a) ITA; (b) k-IMA for $k = 2$.

Corollary 3.4. *Any language accepted by a $T(n)$-time bounded* ITA *or a $T(n)$-time bounded k-*IMA *can be accepted by an* ATM *in time* $O(T(n))$.

Corollary 3.4 is stronger than the result given in [25], where it was shown that $T(n)$-time bounded ITAs and $T(n)$-time bounded k-IMAs can be simulated by $O(T^2(n))$-time bounded ATMs and $O(T^k(n))$-time bounded ATMs, respectively. In fact, Corollary 3.4 holds even for the case of an X-*tree* array, answering a question posed in [25]. (An X-tree array is an ITA in which two-way communication between adjacent nodes on the same level of the tree is allowed.)

A $T(n)$-time bounded ATM can be simulated by an $O(T(n))$-time bounded ITA [25]. The proof is a generalization of the technique in [8] for simulating a nondeterministic TM by an ITA. Thus, from Corollary 3.4, we obtain the following.

Corollary 3.5. *The following are equivalent:* $O(T(n))$*-time bounded* ITAs, $O(T(n))$*-time bounded* X-*tree arrays, and* $O(T(n))$*-time bounded* ATMs.

Corollaries 3.4 and 3.5 together answer in the negative the following question posed in [25]: is there a language accepted by a linear-time k-IMA which requires nonlinear time to accept on an ITA? Our result shows that the answer is no. The question for the case when "linear time" is replaced by "real time" (posed in [8]) remains open.

4. Simulations by space-bounded Turing machines

The results of the previous section immediately give efficient simulations of OWCs by space-bounded TMs. This follows from the fact that a $T(n)$-time bounded ATM can be simulated by a $T(n)$-space bounded TM [2]. Interestingly, by exploiting the regularity of interconnections of some OWCs, better space-bounded TM simulations can be obtained. In particular, we show the following.

Theorem 4.1.

(1) *Any language accepted by a* OWTA *can be accepted by a* TM *in space* $(\log^2 n)/(\log\log n)$.

(2) *Any language accepted by a* OWTRA *can be accepted by a* TM *in space* $n\sqrt{n}$.

(3) *Any language accepted by a $S^k(n)$-processor bounded k-*OWMA *can be accepted by a* TM *in space* $S^{2-1/k}(n)$.

Proof. The proof of Theorem 4.1(1) is given in [17]. We only prove parts (2) and (3) of the theorem.

Proof of Theorem 4.1(2): We first give a straightforward simulation of a OWTRA by a TM that uses space n^2, then show that this can be improved by "partitioning" so as to obtain the desired space bound of $n\sqrt{n}$. Let A be a OWTRA. Label the

nodes of A such that $\langle i, j\rangle$ represents the jth node at depth i, with the input nodes at depth 0 and the nodes at a given depth numbered 1, 2, etc., from left to right. Thus, the accepting node is labelled $\langle n-1, 1\rangle$ and the input nodes are labelled $\langle 0, j\rangle$, $1 \leq j \leq n$. A TM M simulates A by implementing the following recursive algorithm.

Procedure SIMUL($n, a_1 \ldots a_n$);
//Simulates the OWTRA A on input $a_1 \ldots a_n$ by repeatedly calling function STATE to compute the state of the accepting node.//
  $t \leftarrow 0$; $p \leftarrow q_0$;
  repeat
    $p \leftarrow$ STATE($\langle n-1, 1\rangle, t, p$);
    $t \leftarrow t+1$
  until p is an accepting state and $t \geq n$;
end SIMUL.

function STATE($\langle i, j\rangle, t, q$);
//Returns the state of node $\langle i, j\rangle$ at time t, given its state q at time $t-1$. We assume that the state of a node at time $t<0$ is q_0.//
  case
  $i=0$: //an input node//
    if $t=0$ **then** STATE $\leftarrow a_j$ **else** STATE $\leftarrow \delta_0(q)$;
  $i>0$: //a non-input node//
    if $t=0$ **then** STATE $\leftarrow q_0$
    else
      $p_L \leftarrow q_0$;
      for $k \leftarrow 0$ **to** $t-1$ **do** $p_L \leftarrow$ STATE($\langle i-1, j\rangle, k, p_L$);
      $p_R \leftarrow q_0$;
      for $k \leftarrow 0$ **to** $t-1$ **do** $p_R \leftarrow$ STATE($\langle i-1, j+1\rangle, k, p_R$);
      STATE $\leftarrow \delta_2(q, p_L, p_R)$
    endif;
  endcase;
end STATE.

It is easy to see that the depth of recursion of function STATE is at most $n-1$, the depth of A. Also, the space required to store the arguments $(\langle i, j\rangle, t, q)$ is no more than $O(\log n + \log T(n)) = O(\log T(n))$. Therefore, TM M uses $O(n \cdot \log T(n))$ space for the simulation.

The space can be reduced by partitioning the nodes of A into blocks of size $d \times d$, as shown in Fig. 4. Informally, each block is treated as a "supernode" whose "superstate" at time t consists of the d^2 states at time t of the nodes within the block. (In general, some blocks at the bottom level may not have exactly d^2 nodes; however, one can fill these blocks with dummy nodes whose states are permanently set to some state λ.) As before, the superstate of a supernode at time t depends on

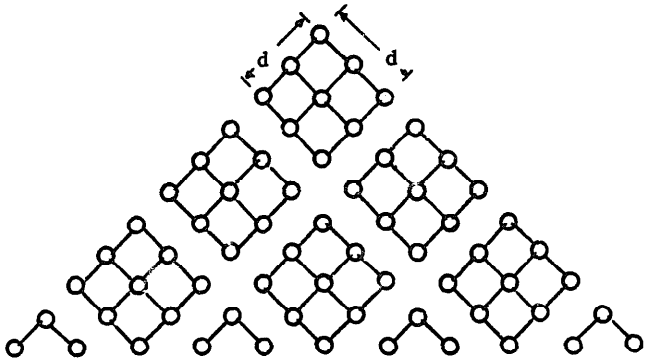


Fig. 4. Partitioning the nodes of a OWTRA.

its superstate at time $t-1$ and the superstates of its two predecessor supernodes at time $t-1$. The simulation by M remains the same, except that the arguments to function STATE are the block indices, time, and superstate, respectively. Since the number of blocks is $O(n^2/d^2)$, the space required to store the arguments is

$$O(\log(n^2/d^2)+\log T(n)+d^2)=O(\log n-\log d+n+d^2),$$

since $T(n)\leq c^n$. Moreover, the depth of recursion of function STATE is $O(n/d)$. The total space used is thus $O(n/d\cdot(n+d^2))$, which is minimum when $d=\sqrt{n}$. This gives a space bound of $O(n\sqrt{n})$ which can be reduced to $n\sqrt{n}$.

Proof of Theorem 4.1(3): The proof for $k=1$ is straightforward since a $S(n)$-processor bounded 1-OWMA can be directly simulated by a $S(n)$-space bounded TM by assigning one cell of the TM worktape for each node of the 1-OWMA. For $k>1$, the simulation is essentially the same as in the proof of part (2), except that the nodes of the k-OWMA are partitioned into k-dimensional blocks of size d to a side. The depth of recursion is $O(S(n)/d)$ and the space needed per recursive level is $O(\log(S^k(n)/d^k)+\log T(n)+d^k)=O(\log S(n)-\log d+S(n)+d^k)$ (since $T(n)\leq c^{S(n)}$). The space required is thus $O(S(n)/d\cdot(S(n)+d^k))$, which is minimum when $d=S^{1/k}(n)$. This gives a space bound of $O(S^{2-1/k}(n))$. □

A consequence of Theorem 4.1 is that two-way communication strictly increases the power of one-way conglomerates. It is known that a TWTA is equivalent to a n-space bounded TM [10]. Similarly, it is easy to show that a TWTRA can simulate a n^2-space bounded TM. (The TWTRA simply propagates the input to its apex then simulates the TM worktape by wrapping it around the diagonals.) Using the same idea, it can be shown that a $S^k(n)$-processor bounded k-TWMA can simulate a $S^k(n)$-space bounded TM. Thus, from Theorem 4.1 and the space hierarchy theorem [14], we have the following.

Corollary 4.1. *There is a language accepted by a* TWTA, TWTRA, *or* $S^k(n)$*-processor bounded* k-TWMA, $k\geq 2$, *that cannot be accepted by a* OWTA, OWTRA *or* $S^k(n)$*-processor bounded* k-OWMA, *respectively.*

It is not known whether a $S(n)$-space bounded 1-TWMA is strictly more powerful than a $S(n)$-space bounded 1-OWMA since Theorem 4.1(3) gives a space-bounded TM simulation for a 1-OWMA which is no better than that for a 1-TWMA (i.e. both require $S(n)$ space). It seems difficult to improve the $S(n)$ space needed to simulate a 1-OWMA in light of the fact that a $S(n)$-processor bounded 1-OWMA can simulate a $S(n)$-time bounded ATM [4]. Thus, an improvement on the space bound would imply that the class of languages accepted by $S(n)$-time bounded ATMs are strictly contained in the class accepted by $S(n)$-space bounded TMs, settling a long-standing open question [2].

5. Conclusion

We have given efficient time-bounded ATM and space-bounded TM simulations of one-way conglomerates, which are interconnected networks of finite-state machines that allow only one-way communication between adjacent nodes. The results remain valid for arrays with outputs (i.e. they become transducers). An interesting extension we have not addressed is when the nodes of the array are not finite-state, e.g. a node can have states, inputs and outputs of $O(\log n)$ bits each. For this case, the simulation time of the ATM given in Theorem 3.1 would seem to require an additional multiplicative factor of $O(\log n)$ because $O(\log n)$ bits have to be guessed at each recursive level. Thus, for example, a non-finite-state OWTRA operating in c^n time would require $O(n^2 \log n)$ time on an ATM, as opposed to $O(n^2)$ for the finite-state case. On the other hand, by slightly modifying the proof of Theorem 4.1(2), it can be shown that a non-finite-state OWTRA can be simulated by a TM in space $n(n \log n)^{1/2}$. The more efficient TM simulation comes from the ability of a TM to "save space" by partitioning, which apparently does not "save time" on an ATM. This may well be one of the few examples in support of the widely held belief that TM space is more powerful than ATM time.

Acknowledgment

We thank Michael C. Loui for sending us a copy of [25] and for bringing [11] to our attention. We also thank Anastasios Vergis for his help in the proof of Lemma 3.1. Finally, we thank the referees for their comments that helped improve the presentation of the paper, and for pointing out that our original version of Theorem 3.1 can be improved by defining uniformity of one-way conglomerates in terms of Ruzzo's extended connection language, instead of the directed connection language [21].

References

[1] M. Atallah and S. Kosaraju, A generalized dictionary machine for VLSI, *IEEE Trans. Comput.* **34**(2) (1985) 151-155.

[2] A. Chandra, D. Kozen and L. Stockmeyer, Alternation, *J. Assoc. Comput. Mach.* **28**(1) (1981) 114-133.

[3] J. Chang, O. Ibarra and M. Palis, Efficient simulations of simple models of parallel computation by space-bounded TMs and time-bounded alternating TMs, Computer Science Department, University of Minnesota, Tech. Rep. TR 85-47, 1985.

[4] J. Chang, O. Ibarra and A. Vergis, On the power of one-way communication, in: *Proc. IEEE 27th Ann. Symp. on Foundations of Computer Science*, Toronto, Canada (1986) 455-464; also *J. Assoc. Comput. Mach.*, **35** (1988) 697-726.

[5] J. Chang, M. Chung, O. Ibarra and K. Rao, Systolic tree implementation of data structures, *Proc. International Conference on Parallel Processing*, St. Charles, IL (1986) 669-671; also *IEEE Trans. Comput.*, **37** (1988) 727-735.

[6] Y. Chiang and K. Fu, Parallel parsing and VLSI implementations for syntactic pattern recognition, *IEEE Trans. Pattern Anal. Mach. Intell.* **6**(3) (1984) 302-313.

[7] S. Cole, Real-time computation by n-dimensional iterative arrays of finite-state machines, *IEEE Trans. Comput.* **18**(4) (1969) 346-365.

[8] K. Culik II and S. Yu, Iterative tree automata, *Theoret. Comput. Sci.* **32** (1984) 227-247.

[9] K. Culik II, O. Ibarra and S. Yu, Iterative tree arrays with logarithmic depth, *Internat. J. Comput. Math.* **20** (1986) 187-204.

[10] C. Dyer and A. Rosenfeld, Triangle cellular automata, *Inform. and Control* **48** (1981) 54-69.

[11] L. Goldschlager, A universal interconnection pattern for parallel computers, *J. Assoc. Comput. Mach.* **29**(4) (1982) 1073-1086.

[12] L. Guibas, H.-T. Kung and C. Thompson, Direct VLSI implementation of combinatorial algorithms, in: *Proc. Caltech Conference on VLSI* (1979) 509-525.

[13] F. Hennie, III., *Iterative Arrays of Logic Circuits* (MIT Press, Cambridge, MA, 1987).

[14] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA, 1979).

[15] O. Ibarra, S. Kim and S. Moran, Sequential machine characterizations of trellis and cellular automata and applications, *SIAM J. Comput.* **14** (1985) 426-447.

[16] O. Ibarra, S. Kim and M. Palis, Designing systolic algorithms using sequential machines, *IEEE Trans. Comput.* **35**(6) (1986) 531-542.

[17] O. Ibarra and T. Jiang, On iterative and cellular tree arrays, *J. Comput. System Sci.* **38**(3) (1989) 452-473.

[18] S. Kosaraju, On some open problems in the theory of cellular automata, *IEEE Trans. Comput.* **23** (1974) 561-565.

[19] S. Kosaraju, Speed of recognition of context-free languages by array automata, *SIAM J. Comput.* **4**(3) (1975) 331-340.

[20] W. Ruzzo, Tree-size bounded alternation, *J. Comput. System Sci.* **21** (1980) 218-235.

[21] W. Ruzzo, On uniform circuit complexity, *J. Comput. System Sci.* **22** (1981) 365-383.

[22] J. Seiferas, Iterative arrays with direct central control, *Acta Inform.* **8** (1977) 177-192.

[23] A. Somani and V. Agarwal, An efficient unsorted VLSI dictionary machine, *IEEE Trans. Comput.* **34**(9) (1985) 841-852.

[24] L. Stockmeyer and U. Vishkin, Simulation of parallel random-access machines by circuits, *SIAM J. Comput.* **13**(2) (1984) 409-422.

[25] J.L. Trahan, Simulations among multidimensional iterative arrays, iterative tree automata, and alternating Turing machines, University of Ilinois, Coordinated Science Laboratory, Technical Report UILU-ENG-86-2202 ACT-66, 1986.